DEPARTMENT OF PHILOSOPHY,
LINGUISTICS AND THEORY OF SCIENCE

FAST VISUAL GROUNDING IN INTERACTION

Bringing few-shot learning with neural networks to an interactive robot

José Miguel Cano Santín

Master's Thesis:	30 credits
Programme:	Master's Programme in Language Technology
Level:	Advanced level
Semester and year:	Spring, 2019
Supervisor	Simon Dobnik, Mehdi Ghanimifard
Examiner	Aarne Ranta
Report number	(number will be provided by the administrators)
Keywords	Grounding, object learning, interactive learning, transfer learning, neural networks

Abstract

A big challenge for the development situated agents is that they need to be capable of grounding real objects of their enviroment to representations with semantic meaning, so they can be comunicated to human agents using the human language. de Graaf (2016) developed the KILLE framework, which is a static camera-based robot capable of learning objects and spatial relations from very few samples using image processing algorithms suitable for learning from few samples. However, this framework has a major shortcoming: the time needed to recognise an object increased greatly as the system learned more objects, which motivates us to design a more efficient object recognition module.

The following project researches a way to improve object recognition of the same robot framework using a neural network approach suitable for learning from very few image samples: Matching Networks (Vinyals et al., 2016). Our work also investigates how transfer learning form large datasets could be used to improve the object recognition performance and to make learning faster, which are very important features for a robot that interacts online with humans.

Therefore, we evaluate the performance of our situated agent with transfer learning from pre-trained models and different conversational strategies with a human tutor. Results show that the robot system is capable of training models really fast and gets very good object recognition performance for small domains.

Preface

It is a fact that my fascination for robotics development has been one of the main reasons why I have choosen to work in this field for my master's thesis in the MLT programme. Robots have always attracted me since I was a child watching and reading all kind of science-fiction and fantasy movies and books. It is not the only one, however, but my choice has also been driven by the interest in developing my knowledge in machine learning and dialogue systems. I have enjoyed working in this project, I have learned a lot from the challenges I had and I have seen how much I have still to learn in the future.

I would like to acknowledge in the first place all the help and support that my supervisors Simon Dobnik and Mehdi Ghanimifard have provided me during all this thesis period. Without their hard work and enthusiasm, I am quite sure this thesis would not be half as good as it is.

I am also very grateful to my dear family and all my amazing friends, no matter if they are living five minutes from my house or in the other side of Europe. Without their all support, their music, the interminable talks and their lovely beings in general I would not have been here for sure.

I would also like to thank all my colleagues in the MLT programme for these two wonderful years we have spent together and all the support we have provided each other and the nice moments we had. We made it, my friends.

Contents

1 Introduction 1

1.1 Visual grounding: lexical semantics 2

2 Background 3

2.1 The KILLE framework 3

2.2 One-shot learning 3

2.3 Interactive strategies for situated learning 4

3 Materials and methods 5

3.1 The robot scenario 5

3.2 Neural network model 7

3.2.1 Image encoder: transfer learning 7

3.2.2 Object classification: matching networks 8

3.3 SOTA dataset 11

3.4 Training strategies: implementing Matching Networks to a robot framework 11

3.4.1 Original training strategy 11

3.4.2 Training strategy in our robot scenario 13

3.5 Learning objects through interaction 15

4 Experiments and discussion 19

4.1 Experiment 1: System validation 19

4.2 Experiment 2: Transfer Learning and Training Strategy 20

4.3 Experiment 3: Baseline evaluation 22

4.4 Experiment 4: Building a support set with images from another domain 23

4.5 Experiment 5: Learning new classes of objects 25

5 Conclusions and further work 27

References 29

6 Appendices 31

6.1 Robot implementation 31

6.1.1 dialogue.py 31

6.1.2 matchingnets.py 31

6.1.3 recognise.py 31
6.2 SOTA dataset collection 32

1 Introduction

Situated agents such as robots, since they are embodied physically, need to engage in dialogue. This includes interacting with objects and, foremost, humans, with whom robots need to be able to dialogue in order to provide a good user experience. To fulfill both requisites, these agents need to ground real world objects and entities that they see with their cameras to human language.

These systems need not only to link entities with language, but to be also capable of continuously learning new ones, that is, to learn new objects while they are interacting with humans (Skočaj et al., 2010). Children are able to learn to ground real objects to their language capabilities, for instance a child can generalise the concept of a *giraffe* from seeing a single picture (Vinyals et al., 2016). It would be also ideal, then, if situated agents were capable of learning from one or very few samples. That is why we introduce as the main contribution of this work the implementation of one-shot learning techniques with neural networks and pre-trained convolutional neural network (CNN) for image encoding in a robot framework.

KILLE (Kinect Is Learning LanguagE) is a robot framework for grounded learning of language developed by de Graaf (2016). Vinyals et al. (2016) introduced Matching Networks: a neural network approach for rapid and efficient one-shot learning. In our work, we develop and test a system that integrates and improves both approaches.

Our implemented neural network aim to address three main research questions. Firstly, we want to see if robots are able to learn objects from very few samples and in a fast way using neural networks. That is to say, to have models which can be perform fast learning and do not require of large collections of samples to be reliable (Vinyals et al., 2016). Secondly, in the same way we aim to investigate how existant external collections of image captions such as ImageNet (Russakovsky et al., 2015) or MSCOCO (Lin et al., 2014) can be utilised to pre-train models and how to transfer and take advantage of the knowledge of this pre-training to make easier for a robot system to learn to recognise the objects perceived by its camera. Finally, we want to research how interaction of a situated agent with a human tutor can make object learning more efficient.

The first two research questions have been raised in the master's thesis by de Graaf (2016), who developed the setup of the robot that we use for this project and its first implementation in an interactive scenario with object recognition and spatial relations grounding. We take the development of the KILLE robot framework and explore to what degree few-shot neural network techniques can be used in a situated agent framework.

An additional contribution of this project is the creation of a small dataset of images taken with our camera that we haved called SOTA (Small Objects daTAset). This small sized dataset currently consists of 400 images distributed equally into 20 categories of objects, which makes it suitable for few-shot learning tasks. In this project, it is used to train and evaluate our robot neural network models. We have decided to publish it as it can be used for further improvement of this robot framework and it could be ported to any other image recognition system.

The following thesis is divided into five sections, including this introduction to the project. Section 2 provides an explanation of the works that serve as background for this one. Section 3 describes the hardware setup of our robot and the different modules that constitute its software. Section 4 contains a description of experiments, results and discussion. Finally, Section 5 summarises our work and suggests possible future improvements for this robot framework.

A short paper directly related to this work has been published in The 23rd Workshop on the Semantics and Pragmatics of Dialogue (SemDial) in September 2019 (Cano et al., 2019).

1.1 Visual grounding: lexical semantics

Interactive robots such as the one in this work need to ground the information about their enviroment, which they perceive with their sensors, to meaningful symbols (e.g. visual and auditive information) such as humans connect their enviroment to language symbols.

Harnad (1990) defines grounding as connecting symbolic descriptions of the real world to semantically interpretable symbol descriptions: object and entity names (de Graaf, 2016). How humans connect language with spacial perception and actions has been studied from both linguistics (lexical semantics) and psychology (perception and cognition). Understanding how language and vision connects can help us to understand how humans ground entities, spatial relations and actions into their cognition and, consequently, how technology can benefit from it.

Visual grounding is also of interest also for computational systems (language modelling) and robotics as well. That is, developing "trainable systems that can learn domain specific rules of language generation from examples" (Roy, 2002), which could be images, video, sound, etc. A situated agent which can interact with the world and ground its enviroment into human language is a very useful tool.

For instance, navigation systems providing directions, working robots developing tasks or interactive toys for kids can benefit a lot of that capacity of interacting with their enviroment and being able to communicate to humans what they are actually perceiving. However, to have this ability to communicate they need to be capable first of grounding what they see into human language.

Such agents also need to deal with a constantly changing world. From a visual grounding perspective: unseen objects and entities, differents shapes and volumes, movemement (related to spatial relations) and the same language change are a few examples of new inputs which the robot needs to be able to deal with.

Our robot system is situated in the field of visual grounding as we have developed a system that uses neural network techniques to ground the objects that sorround the robot (e.g. an apple, a shoe or a chair) into their semantic symbols. Therefore, our situated agent can communicate to a human what it is seeing at that moment.

It is important to consider as well how humans call objects and, therefore, how visual grounding models should call them (Ordonez et al., 2013). In the same way language symbols may be connected to each other in some way. That is, "object categories form a semantic hierarchy consisting of many levels of abstraction" (Deng et al., 2012).

For instance, a *boot* is partially a *shoe*, and both objects share visual and also conceptual characteristics. In the same way, although you can say *horse*, a horse is also an *equine*, a *mammal*, an *animal*, etc. In both examples all the possible subcategories share a lot of features that makes them be part of a supercategory, but conceptually humans differentiate them using characteristics which are not always visual (e.g. texture, flavour or how can you interact with it). Although this is outside the range of our work, it is also worth considering how humans conceptualise what they perceive and how they ground it in language by relating new objects to what they already know.

2 Background

2.1 The KILLE framework

The starting idea and basis of our robot setup is the KILLE (Kinect Is Learning LanguagE) system (de Graaf, 2016). This system consists of ”a non-mobile table-top robot connecting Kinect sensors with image processing and classification and a spoken dialogue system” (Dobnik & de Graaf, 2017a, p. 1). Our project uses the same framework in investiagtion of a new research question, namely the one-shot learning with neural networks.

In the original KILLE system, object recognition was based on fixed features processed with the SIFT (Scale-Invariant Feature Transform) algorithm (Lowe, 1999), which was chosen because it provided good recognition perfomance with few samples (de Graaf, 2016). The SIFT algorithm works by selecting keypoints from smoothed images taken with the camera of the robot. Those keypoints represent sudden changes in colour (e.g. line and corner detection). Those SIFT keypoints are then stored in a database and compared with the current frame of the camera when the system is asked to recognise an object.

However, one of the main shortcomings of the SIFT algorithm that (de Graaf, 2016) remarks is that the computing time to recognise an object increases a lot when more objects and categories are added, since it has to compare the current frame of the camera with more images. Therefore, (de Graaf, 2016) remarks two possible improvements to its system related to replace the SIFT algorithm with neural network models.

In the first place, replacing the features learned with SIFT with convolutional neural network (CNN) encoders would solve the issues in time performance with more image samples. CNNs learn the features of the images from scratch, but once these are learned after training a CNN model for image encoding, they can be quickly applied to encode an image as a vector of a fixed size.

In the second place, neural network models could also allow another improvement: transfer learning (Dobnik & de Graaf, 2017a). That is, pre-training a model offline on a large corpus and use the knowledge about encoding the relevant image features that has learned in this pre-training in our robot system online.

These two points will be further explained in Section 3.

2.2 One-shot learning

Deep learning techniques have provided very important advances in the area of language and vision, specially during the last ten years. For instance, very deep convolutional neural networks achieved significant improvement in image classification (Simonyan & Zisserman, 2014). However, neural networks are also ”notorious for requiring large datasets” (Vinyals et al., 2016, p. 1) and, consequently, also taking very long time to train effective models.

According to (Cai et al., 2018), research on one-shot learning techniques has taken mainly four different paths. Probably the most evident one is data augmentation, which aims to address the issue of having few data by applying automatic creation of images transformed from the ones in the dataset. That is, generating more images from the original ones by applying one or more transformations such as vertical and horitzontal translation, rotation or color and contrast changes (Dosovitskiy et al., 2014). Data augmentation alleviate overfitting with few data, but does not solve the problem of needing long training times (Vinyals et al., 2016).

Secondly, transfer learning aims to reuse ”the knowledge learned from previous tasks for one-shot learning”

(Cai et al., 2018, p. 2). For instance, an image classification model pre-trained on a large dataset of images could help to learn novel categories from very few instances of those categories by reusing the same model, either by using the same model or fine-tuning on the images from the new category (Wang & Hebert, 2016).

Thirdly, research on few-shot learning has also attempted to "create a low-dimensional embedding space, where the transformed representations are more discriminative" (Cai et al., 2018, p. 2). The embedding spaces built in these techniques are quite close to the ones built in nearest neighbour machine learning techniques (Vinyals et al., 2016) such as the KNN algorithm. They are low-dimensional in terms of the images features they work with, and therefore they are suitable for one-shot learning scenarios. After building these embedding spaces, these algorithms classify images by applying comparison methods such as nearest neighbour rules (Koch et al., 2015) or matching with cosine similarity (Vinyals et al., 2016).

Finally, meta-learning aims to acquire continuous knowledge across all tasks that are trained. That is, it approaches rapid learning by acquiring knowledge "within a task, for example, when learning to accurately classify within a particular dataset. This learning is guided by knowledge accrued more gradually across tasks" (Santoro et al., 2016, p. 1-2). That is to say, the system is learning how to learn for new tasks.

In our research we focus on two of these directions. Firstly, we use transfer learning so our robot object recognition system can take advantage of image classifiers pre-trained on large datasets such as ImageNet (Russakovsky et al., 2015) to classify the objects in our robot domain. Secondly, we base our image classification neural network on a low-dimensional embedding space algorithm: the Matching Networks (Vinyals et al., 2016).

2.3 Interactive strategies for situated learning

Situated robotic agents need to continuously acquire new knowledge to ground the objects of their sorrounding enviroment so they can be described to humans. Such knowledge could be acquired on their own (e.g. by training an image classification model). Another and maybe more effective strategy is to learn interacting with other situated agents or humans (Skočaj et al., 2010).

Skočaj et al. (2009) proposed a systematic way of classifying learning strategies of a robot interacting with a human tutor. Their strategies assume the existance of a learning algorithm in the dialogue agent (for instance, our image classification model with neural networks), but they are agnostic to which one is being used as long as it fulfills certain basic requisites. Firstly, it needs to be *incremental*, so must be able to acquire new knowledge constantly. The system should be able to *unlearn* so it can forget or correct wrongly classified samples or treat them as negative examples. Finally, it is supposed to have certain level of *self-understanding* so it is able to "estimate whether its current knowledge suffices to interpret the current scene, or it should ask the tutor for help" (Skočaj et al., 2009, p. 3).

Dialogue strategies in our robot implementation focus on strategies for acquiring knowledge incrementally and self-understanding, but other such as unlearning could be also implemented in the Python script that controls the interaction strategies (Section 3.5).

The Kille framework (Dobnik & de Graaf, 2017a) also worked on implementing interaction strategies for situated agents with a human tutor. Their dialogues use two main user-initiated strategies for learning objects and spatial relations incrementally: direct instructions (*This is a cup*, *The book is to the right of the mug*) and querying about an object which is being observed (*What is this?*). They also implemented the capability to make the system unlearn objects (*Forget a book*, *That is not what I said*).

Regarding the capability of the robots to *unlearn*, the work by Schütte et al. (2015) focuses on reformulation strategies for referring expressions. These strategies are able to deal with situations in which the robot

has provided incorrect references to the physical objects. For example, when a robot classifies an object incorrectly or provides a wrong spatial relation (e.g. saying that an object is to the left when it is above).

3 Materials and methods

3.1 The robot scenario

Our situated agent setup is based on the KILLE framework (de Graaf, 2016). This setup consists of a stationary Microsoft's Kinect v1 RGB-D (red, green, blue and depth) sensor (Figure 1) connected to an Ubuntu 16.04 system. The sensor is supported by the *Freenect* drivers[1] integrated in the Robot Operating System (ROS) framework (Quigley et al., 2009), which allows Kinect to be used on any personal computer. However, since it is an old version of Kinect and drivers have not been updated since the ROS Kinetic version, this camera hardware does not support Ubuntu versions newer than 16.04.

In front of this stationary sensor a small object is presented for the system to recognise, as shown in Figure 2. This system lacks of any mobility by itself, but it could be extended with other modules that control movement or other RGB sensors. This is due to the fact that our implementation just controls the RGB sensor through ROS, the object recognition and the human interaction (dialogue) with the robot. Therefore, it could potentially be implemented as another module in a movable robot which uses ROS, as far as it has an RGB camera. This potential portability to other hardware systems is one of the main benefits of using the ROS framework for this project.

This project does not use the camera depth sensor, but only the RGB one. The KILLE robot (de Graaf, 2016) uses the depth sensor to remove the background (everything more than 100 cm from the camera position) and foreground (less than 70 cm, which is also the minimum distance of the depth sensor for perceiving objects) of the images taken by the camera, so just the object appears on the final image with a white background, as shown in Figure 3. This is an easy way to get attention over the objects, so the background does not interfere in the recognition process.

However, using the depth sensor filter has some shortcomings. Firstly, since the the laser emitter and the camera of the depth sensor are separated about 7 cm from each other (Figure 1), the infrared image which represents the depth and the RGB image have a small deviation. Consequently, the final image with the removed background tends to cut one side of the objects. This issue can be addressed by moving the background filter image a few centimeters to get acceptable results as in Figure 3, but it is not perfect and the cutting is a lot worse with smaller objects.

Another issue of the background filtering related to the size of the objects is given by the limitation on the distance at which the objects can be relative to the camera. Since we can only place objects at 70 to 100 cm from the camera, this makes very small objects (e.g. a pen or a clothespin) very difficult to recognise as they are seen from far. Big objects (e.g. a chair or a big box) would be also impossible to be seen entirely by the camera with the background filtering. This background filtering distance can be changed since the depth sensor can perceive objects up to 300 centimeters of the camera, but extending it would also make more difficult to separate the object of interest from the background objects in certain enviroments (e.g. small rooms).

Our implementation is not making use of the background filtering because of the shortcomings detailed above, specially the problems that has with seeing small objects from too far, which combined with the depth sensor deviation makes almost impossible for the camera to see these small objects. However, the depth sensor filter is implemented in our system but deactivated, so it would only take editing some few lines to enable it.

[1] https://openkinect.org/

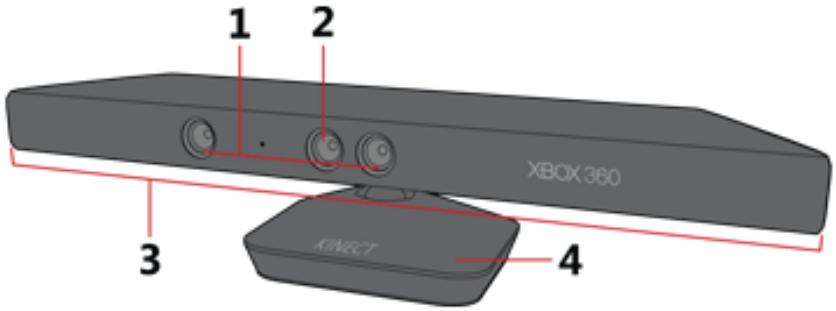

Figure 1: The Microsoft's Kinect v1 RGB-D sensor[2]. (1) is the 3D depth sensor which consists of the infrared laser projector (left) and the infrared camera (right). (2) is the RGB sensor, (3) is a microphone array and (4) is a tilt motor.

Figure 2: The robot setup (left) and what the robot is seeing with its RGB sensor (right).

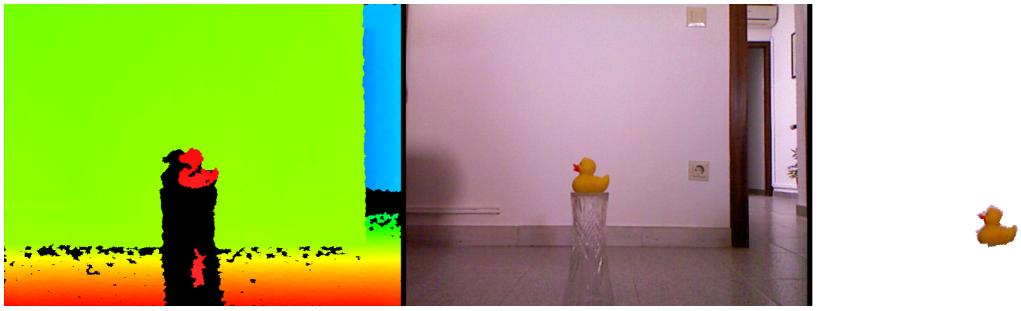

Figure 3: The input from the depth sensor (left), the input from the RGB camera (center) and the RGB with the filtered background (right). Since the glass support is not detected by the camera, only the object appears in the final image.

[2]Image from: https://support.xbox.com/en-US/xbox-360/accessories/kinect-sensor-components

Our implementation consists of three Python scripts which act as different modules: one (`dialogue.py`) controls the dialogue management; the second (`matchingnets.py`) contains the neural network model for online training and some functions to collect and encode the images that will be used in that session; and the last one (`recognise.py`) acts mainly as a bridge that communicates the actions prompted by the dialogue management and runs the neural network model for object recognition. For processing the images inputs from Kinect to be able to use them on the Python script, we use the Open Source Computer Vision (OpenCV) library (Bradski & Kaehler, 2008) , "which is a popular library for computer vision including real-time machine learning applications" (Dobnik & de Graaf, 2017b, p. 3).

We have set up a public GitHub[3] repository so anyone can have access to the code of this project and the published dataset (for the dataset, see Section 3.3).

3.2 Neural network model

Our deep neural network model is implemented inside the script `matchingnets.py` and it consists of two independent modules: a deep convolutional image encoder and the image classification layers.

3.2.1 Image encoder: transfer learning

For the image encoding module, we utilise the VGG16 (Simonyan & Zisserman, 2014), one of the first very deep convolutional neural network (CNN) architectures. As they demonstrate, these deep CNNs are very accurate for "classification and localisation tasks" (Simonyan & Zisserman, 2014, p. 1). VGG16 encoding layers consist of stacks of convolutional layers which "use filters with a very small receptive field: 3 x 3", followed by max-pooling layers (Simonyan & Zisserman, 2014, p. 2), as shown in Figure 4.

In our system we use VGG16 as it is implemented in Keras[4], a library for machine learning with neural networks for Python. The same library has available for downloading and using a VGG16 model already pre-trained on the full ImageNet dataset (Russakovsky et al., 2015). For this project, we are only interested in using the CNN and max-pooling layers pre-trained for image encoding. That is, we only use the layers which encode the visual features of the images which are then applied in the second module of our neural network (Section 3.2.2).

One of the contributions of this project is the usage of learning transferred from a large dataset of images for classification such as ImageNet (Russakovsky et al., 2015) in a robot framework. That is, we use the CNN layers of the VGG16 model specified above so we do not need to train a new CNN model for image encoding, because we can reuse the knowledge that the model has learned from being trained on a large dataset to encode the images that our robot is using. The question we try to answer here is if this background knowledge trained on pictures taken by humans would be useful for the system to discriminate objects selected in our scenario.

(Yosinski et al., 2014) points out that the benefit of using a pre-trained networks decreases when the task or the data employed in the pre-training stage is very different from the target task, although they found that initializing a target task even with "features transferred from distant tasks are better than random weights" and that transferred features help improve performance "even after substantial fine-tuning on a new task, which could be a generally useful technique for improving deep neural network performance" (Yosinski et al., 2014, p. 8).

[3] https://github.com/jcanosan/Interactive-robot-with-neural-networks
[4] https://keras.io/applications/#vgg16

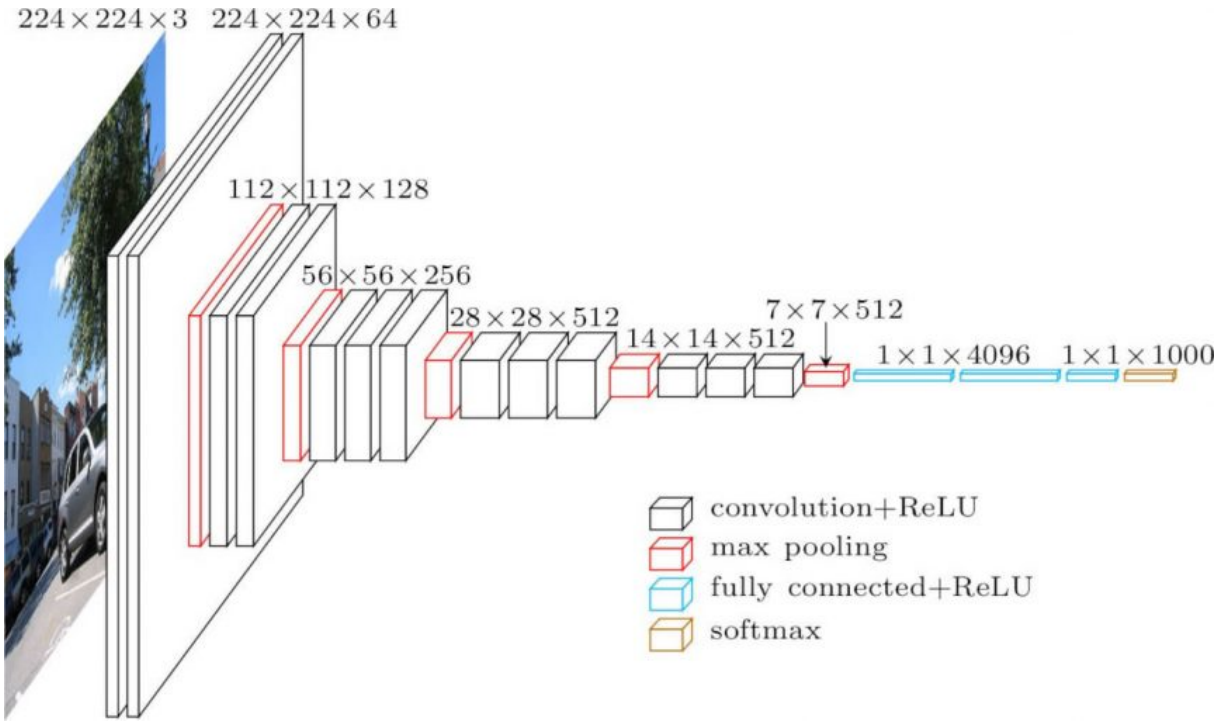

Figure 4: The complete VGG16 neural network[5]. In our implementation we only use the image encoder layers (black and red ones) pre-trained on ImageNet.

The task of both the pre-trained VGG16 model and the neural network that we use in our robot is the classification of a target image. In the same way, both the image data of ImageNet and the images taken by our robot are similar in a way that all of them are prepared for singe object classification, even if they reflect different objects and situations. Therefore, since the task is the same and the data is quite similar, the impact of the transferred knowledge in our system is expected to be positive.

In Section 4 we will discuss the impact on our system performance of using this knowledge transferred from a large dataset.

3.2.2 Object classification: matching networks

Matching Networks (Vinyals et al., 2016) is a neural network algorithm designed for one-shot learning and image and text classification. Its principal advantage is the capability to rapidly learn objects from few observations. That is to say, it is an image classification algorithm which can be trained on one or few image samples and achieve reliable performance (Vinyals et al., 2016).

As the main contribution of our work, we present our implementation of Matching Networks in a robot scenario with interactive grounding. This algorithm has been evaluated in standard image classification sets (Omniglot (Lake et al., 2015) and ImageNet (Russakovsky et al., 2015)) to compare its performance with different baseline algorithms for image classification (Vinyals et al., 2016). As its capacity to learn visual objects fast from very few samples is demonstrated as reliable in the original paper, we considered this algorithm as a possible candidate to be ported to a robot scenario in online interaction with a human tutor.

[5]Image from: https://neurohive.io/en/popular-networks/vgg16/

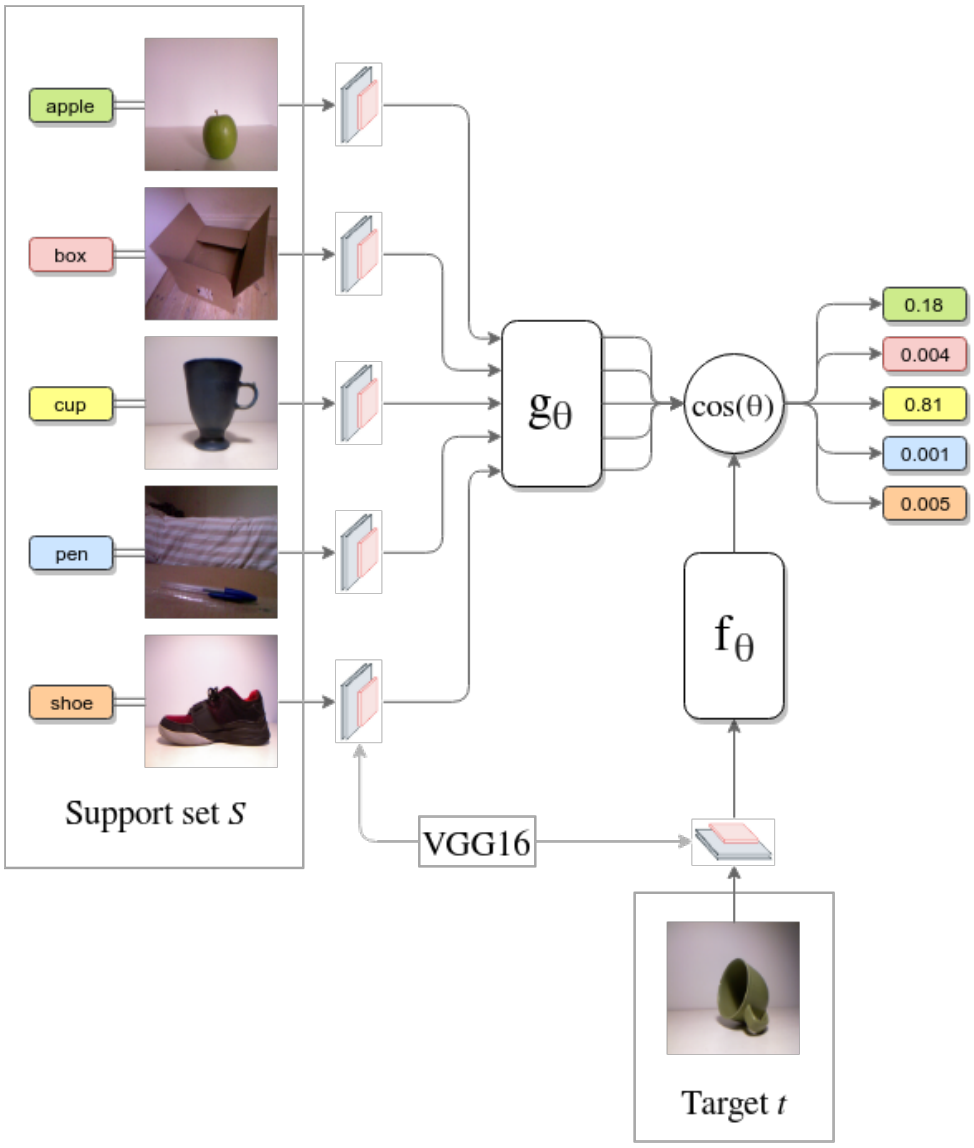

Figure 5: Example diagram of our neural network model for a 1-shot learning task of 5 labelled categories. The five labelled images of the support set (S) are encoded by the VGG16 convolutional layers and the embeddings processed by the g function to learn the dependency of the features between the images of S. In a similar way, the unlabelled target image (t) is also encoded and embedded by its own function f. With both embedding functions, the matching network computes the cosine similarity between the t and the different images of S and outputs the similarity scores for the different labels of S. That is, how similar is t with the different categories. In the example above, we can see that the output scores reflect that t has a lot more similarity with the image labelled as *cup* in S, which makes this an example of a correct classification.

Our matching network version is implemented through TensorFlow[6], a Python library for machine learning and dataflow. We could have used Keras as we did in the image encoder module because it is more simple to build a neural network in that package. However, TensorFlow has two main advantages identified during the first steps of this project: it is more customizable and it performs faster.

As shown in Figure 5, the Matching Networks algorithm works by comparing an unlabelled target image (t) with the labelled images of a support set (S) to determine how similar is t to the images of the different

[6]https://www.tensorflow.org/

categories of S. It is one-shot since the Matching Networks are able to learn the features of the different categories of S from even one single image sample per category, although we will discuss in Section 4 if it is reliable enough with one image per category or if we need more.

In each training instance our model takes a support set S with k labelled images (each one with a size of 224x224 pixels) of each of the n categories of objects and a target image t, which is not labelled but pertains to one of the categories of S. All the images are encoded through the VGG16 module described earlier (Section3.2.1 and passed to our matching network module.

The first main difference between the (Vinyals et al., 2016) Matching Networks and our own adaptation is that the original one does not treat the image encoding layers as a separated module, but as a part of the entire matching network algorithm. Our modular setup has the advantage of allowing us to train each module independently of each other, which allows us to use pre-trained VGG16 as we described in Section 3.2.1, instead of having to train the CNNs along with the matching networks. It also allows our implementation to potentially use any image encoder algorithm with our matching network module.

The input of this module takes the encoded images of S and t and performs two operations on these images. Firstly, it learns the dependency of the features between the images and classes of the entire S set through the embedding function g. In the same way, the target image t is also embedded throught its own function f.

In these embedding functions our model presents significant differences compared to the original work. (Vinyals et al., 2016) use a Bi-directional Long-Short Term Memory (BiLSTM) as the g function to learn those dependencies of the images and categories of S, and for t a normal Long-Short Term Memory (LSTM) as the f function with "read-attention over the whole set S" (Vinyals et al., 2016, p. 3).

Both LSTM and BiLSTM assume a specific order over the categories of the S set which is not discussed in the paper. LSTMs encode ordered sequences and this is beneficial to get dependencies in texts, in which order matters as well. However, for an image classification task that does not seem important since the categories and images do not seem to need any specific order to be classified. Therefore, we thought that a more simple solution would be enough to get reliable results, so we only concatenate the images vectors to learn the embedding functions, but the order of categories is random.

After obtaining the output from the embedding functions, the model computes the cosine similarity of the features of the target t and the features of the images in S. By computing the cosine similarity, the model is able to learn the relevant dependencies of features of S (g) and t (f) when training and, therefore, making the cosine similarity function more effective in further epochs.

Once the cosine similarities have been calculated, similarity scores are computed over the n categories so we get one score per each of the categories in S. These results are computed through a Softmax function so they are normalised and may be presented as a probability distribution in which all the scores add to 1. These scores give us the likelihoods of the target t belonging to the categories of the images in S.

3.3 SOTA dataset

One of the contributions of our work is the creation and publication of the SOTA image dataset (Small Objects daTAset). This consists of a small collection of 400 images with a size of 224x224 each, which is the image size that VGG16 uses by default. As shown in Figure 6, each image depicts a single small object with more or less details in the background, so we have more varied situations in our domain. In general, since we are not using the background filtering in our implementation, it is intended that the object of interest takes a good amount of space in the image, which benefits specially the smaller objects.

Figure 6: Five samples from different categories of the SOTA dataset.

The images in SOTA are distributed equally over 20 categories: *apple*, *banana*, *book*, *boot*, *bottle*, *box*, *clothespin*, *cup*, *desk lamp*, *glass*, *laptop*, *marker*, *pen*, *rubber duck*, *screwdriver*, *shoe*, *spoon*, *stuffed toy*, *thermos* and *toilet tissue*.

As images are classified in categories, this dataset is suitable for evaluation of few-shot learning tasks. However, it could be expanded or utilised alongside with a larger corpus for single object classification tasks.

As stated in Section 3.1 the SOTA dataset is published in the GitHub repository for this project.

3.4 Training strategies: implementing Matching Networks to a robot framework

Matching Networks are capable of learning objects by being trained fast on very few image samples (Vinyals et al., 2016). However, one of the main potential shortcomings that we saw earlier when studying the original Matching Networks is that is that they need to be previously trained, including the CNNs. For this reason, we have chosen to separate the image encoding layers from the matching network layers, as explained in Section 3.2.2.

This makes our implementation a lot faster faster when learning objects online since we do not need to train the VGG16 encoder. Instead, encoding the images with the pre-trained VGG16 module takes about 2.5 seconds for 200 images, as demonstrated in the experiments in Sections 4.2 and 4.3. However, we have to point out that those times have been measured when running the neural network on a graphics card (Nvidia 1050Ti), but we have also observed that it takes longer to encode the images when running on a CPU (about 20 to 25 seconds for the same number of images). To avoid these longer encoding times when interacting online with a robot, in our setup we encode the images once when starting the robot interaction process and then we save the encoded images. Therefore, we do not need to encode again all the images of the support set incrementally to retrain our matching network module in the same session, just the new images that come from the camera when interacting, which are encoded almost instanly since they are never more that one at the same time.

In this section we will detail and compare the two different interaction strategies: the original strategy used in (Vinyals et al., 2016) and the one designed for this robot.

3.4.1 Original training strategy

In the original paper, Vinyals et al. (2016) designed a training strategy for their Matching Networks in which they pre-train the neural network on different small batches of n categories with k images each, and then they use that pre-trained model to classify images on a new batch of different categories (although they need to have the always the same number of categories and images), effectively learning new categories using knowledge transferred from other categories.

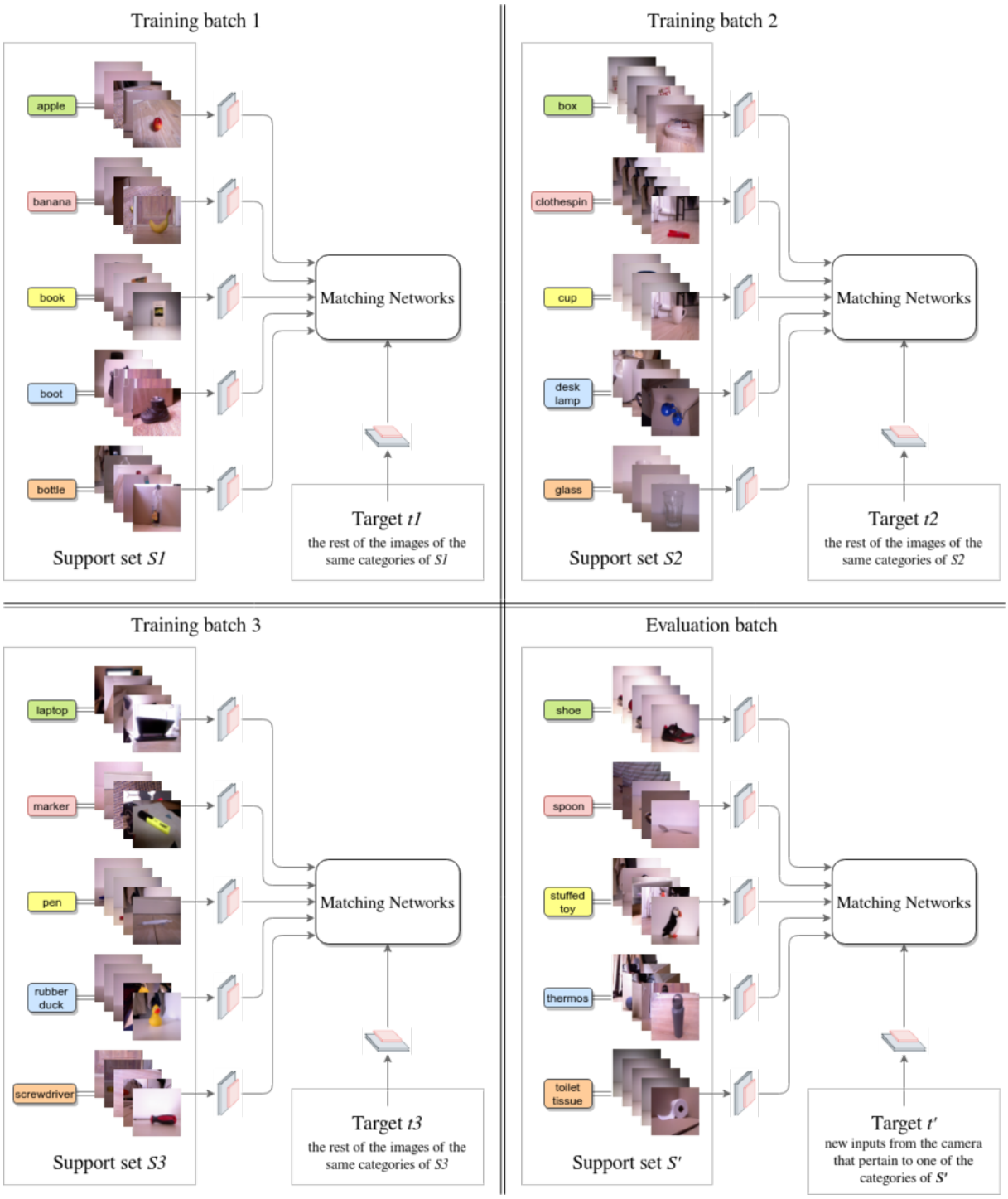

Figure 7: Training strategy for the matching networks as it is in (Vinyals et al., 2016). We train the matching networks on three different batches of labels and use another one with different labels for evaluation on the new inputs of the robot.

This is more clearly explained with an example, which is illustrated in Figure 7. If we want to train a model using their strategy for classifying five categories we should take a dataset, for instance our SOTA dataset. Then we would separate the 20 categories of SOTA into batches of five categories and take k images per category (e.g. one image) and train the model using the first five categories. That is, we train a model on a support set (*S1*) of five categories with one image per category. However, instead of stopping here, we can continue training the same model also on other two batches of five categories (*S2* and *S3*), having then a

model which has been trained on three different batches of categories. The target images (t) for the different training rounds are the rest of the images of the same categories labelled (in this example, 15 images per category since the SOTA categories have 20 images each).

- 1st batch (*S1*, *t1*): *apple*, *banana*, *book*, *boot* and *bottle*.
- 2nd batch (*S2*, *t2*): *box*, *clothespin*, *cup*, *desk lamp* and *glass*.
- 3rd batch (*S3*, *t3*): *laptop*, *marker*, *pen*, *rubber duck*, *screwdriver*.
- Evaluation batch (S', t'): *shoe*, *spoon*, *stuffed toy*, *thermos* and *toilet tissue*.

Instead of using the last of the batches (S') to continue training the model, it is used for classification. That is to say, we take the model pre-trained on the first three batches, build an S' with the test batch (it needs to be of the same size of the other support sets used for training the model) and use this S' to classify unlabelled target images that pertain to the four batch. Therefore, we are not training the model again on the test batch S', but using the knowledge acquired from *S1*, *S2* and *S3* to build the embeddings of S' directly when we try to recognise a target image that pertains to one of the categories of S'.

Still, we need to point out that it is necessary a dataset much larger than SOTA to train effective models with this level of transferred learning. That is, we are transferring knowledge totally from totally different categories as we do in the VGG16 module, which is pre-trained on ImageNet.

The training strategy of (Vinyals et al. 2016) implies that we need different models for each possible size of the support set S. In a trained matching network model the size of S is fixed, meaning that no more categories or images can be added to that model. For instance, if we train a model for an S of five categories and one image per category, we cannot use this model to classify with an S of six categories or three images per category. We would need to pre-train another matching network model on that new size of S to have this option.

3.4.2 Training strategy in our robot scenario

For simplicity and to avoid pre-training a lot of models on large datasets, which would take a lot of time, in our robot setup we use a simpler strategy, but faster to train and reliable enough to evaluate the performance of the matching network for our robot scenario. Also, this training strategy does not require a big dataset of images to perform reliably.

We have to take into account that the strategy used in (Vinyals et al., 2016) is also training the CNN encoding layers along with the Matching Networks, which increases training time. As we are using the CNN layers of a pre-trained VGG16 model with the learning transferred from a large corpus, we are able to skip the training time of the CNNs when learning objects online with the robot and then we only need to train the matching network module, which is more convenient in a robot scenario since it makes possible to train the model online while interacting because it is fast enough, as we will see in Section 4.

We have divided our strategy in two phases: training and prediction. The two phases are illustrated in Figure 8.

In the training phase, we build a support set S with a few (k) images of each labelled class (n). Then, we build a target set t with the same images that we used to build S, and use it as a target to train our model. Then, we are using S as both support set and target in the training phase.

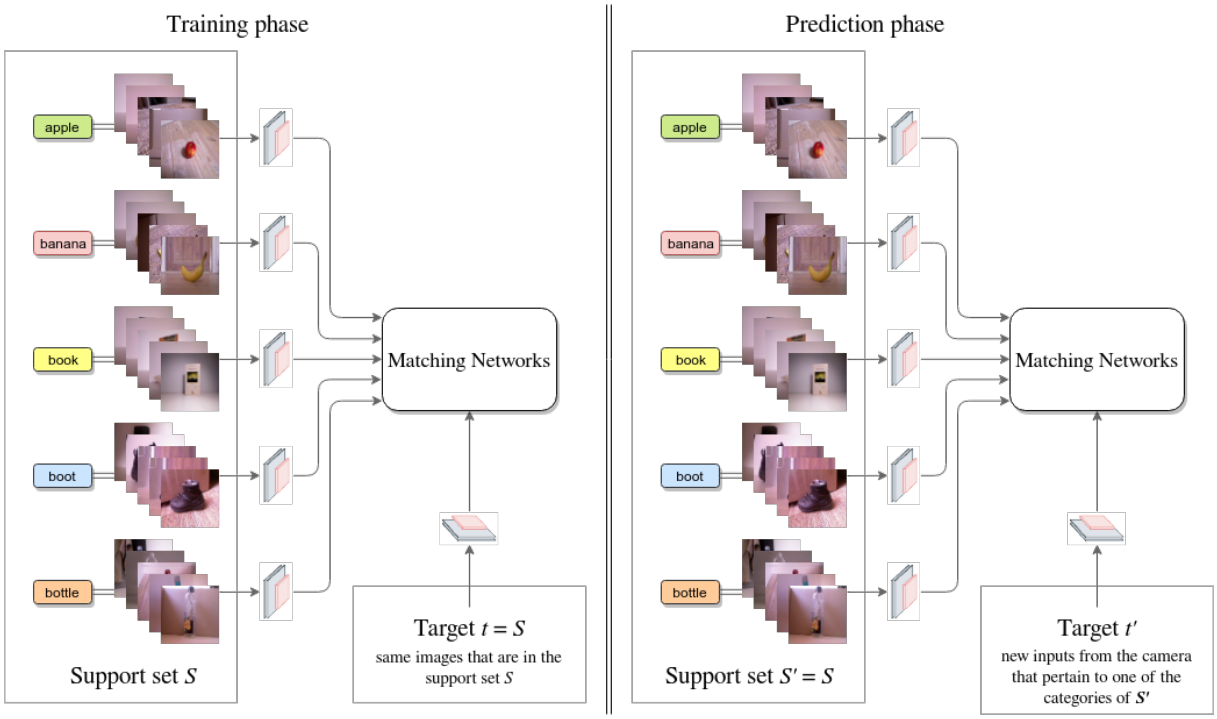

Figure 8: Training strategy for the matching networks in our robot scenario. We train the matching networks with the same support set (S) that we use for the prediction phase.

In the prediction phase, we use as support set the same images that we used as S for training, and classify the target t, which is a new input from the camera taken when the user requests the system to recognise an object which should belong to one of the categories of S.

This training of the matching network is performed online every time the robot needs to *learn a new category* or *update* its knowledge of one of the already learned categories.

To *learn a new category*, the system takes the images taken from the camera of the new category and includes them in the S set. For instance, if we had a model trained on an S of five categories and five images per category and the robot is trying to learn a new category, it would add five images of the new category to S and train a new model on the new S which now has of six categories. If there are not enough images in the dataset of images taken by the camera to learn a new category (that is, at least five images), the robot will ask the user for more samples of that category instead of learning it, as it is explained in Section 3.5.

To *update* (or improve) the knowledge of an already known category, the system will include in the S set the new image taken by the camera and train the model with the new S. This update of S is performed when the user presents an object to the robot, when it assigns an object the wrong category when it tries to recognise the object or when the robot is not confident enough that an objects pertains to a category, even if it has recognised the category of the object correctly. This is further explained in Section 3.5.

With few images and categories training is really fast, as demonstrated in experiments (Section 4). However, as S grows, so does training time. In interactive learning with a robot, it is expected that the system would need to constantly learn new objects and categories. Therefore, having to wait for retraining the model every time we change the size of S is not good from a user perspective.

In further work, a possible way to avoid that could be to configure the system so it only trains a model when the user is not interacting. However, a more effective way of skipping the time that takes the online training

phase would be implementing transfer to different categories, as it is done in the strategy by Vinyals et al. (2016) explained above. That is, splitting some training offline on a large dataset and then doing recognition comparing a new target t image taken with the camera and a support set S built with the images of the objects that were taken earlier with the camera. This S would be the objects that the robot already knows, and the t image the object that is trying to recognise (which should belong to one if the categories in S). With effectively pre-trained matching network models in large datasets, we could skip the online training phase and take full advantage of that transferred learning in our neural network.

We will compare and discuss the impact of using both training strategies for our matching network in Section 4.2.

3.5 Learning objects through interaction

Robotic systems need to acquire constantly new knowledge about their environment and the objects present in it. In grounding, interacting with a human tutor can benefit a system a lot because the conversational partner is able to control *how* the information is presented to the system by their context (Skočaj et al., 2010). Also, the human can control *when* the system should improve. In the context of our situated system, that is *how* the support dataset should be built and *when* the matching network should be trained to improve object recognition.

Dialogue management in our robot is implemented in a very simple way in the Python script `dialogue.py`. It supports text inputs and outputs and it is state-based, as shown in Figures 9 and 10. User inputs are controled by Python default *raw input* functions, while system outputs are Python prints on screen.

The KILLE robot (de Graaf, 2016) used for dialogue management the *OpenDial* toolkit (Lison, 2014), a Java voice-based free and open dialogue management system connected to ROS through a custom module called *ROSDial*. This library has not been implemented in our work as we do not require a sophistication of a full dialogue manager to implement in our experimental system. However, the dialogue rules we have could be integrated with the dialogue management in a complete system.

Our situated dialogue system uses two major dialogue strategies to learn objects, which are based on the original KILLE framework (de Graaf, 2016).

Firstly, Figure 9 shows that the human tutor can present the object (e.g. *This is an apple*), in which case it saves the image of the mentioned object in the dataset. After that, the system checks if there are enough instances of that category in the stored image dataset to train the system reliably.

If the number of images equals the threshold of five images (this chosen threshold will be further discussed in the experiments of Section 4), the system *learns the new category* (e.g. *I am learning apple*), or if there are more than five images it *updates* its knowledge of the object (e.g. *I am updating my systems on apple*), as explained earlier in Section 3.4. If there are less than 5 images in the dataset, the system requests the human tutor to show it more instances about that category (e.g. *Please, show me more examples of apple*).

Secondly, the robot can also handle that the user queries it about an object, as shown in Figure 10 (*What is this?*).

In this case, the robot attempts to recognise the object presented and answer the question. Depending on the certainty of this guess (that is to say, the classification confidence score retrieved when recognising the target image from the camera with the matching network), the system prompts one between four possible answers. Such answer strategies for continuous learning of a robot are based on the work of (Skočaj et al., 2009), who propose dialogue strategies which depend on the recognition confidence score that can be used

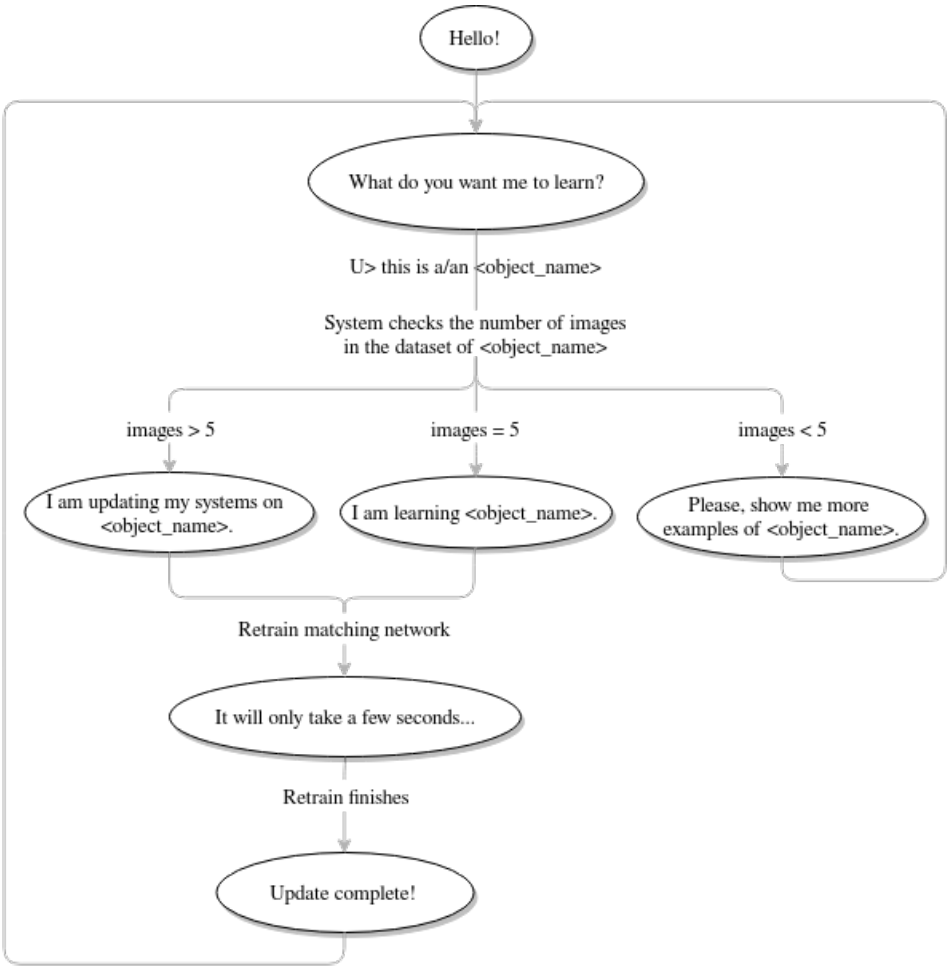

Figure 9: Interaction diagram for when the user presents an object (*This is a/an [...]*). User utterances are marked with *U>*.

on object classification tasks (see Section 2.3).

Based on that proposal, we have devised four possible system answers to the user-initiated query based on the confidence score of the category with the highest score. If that score is equal or higher than 0.8, such recognition has a very high confidence, and therefore the system answer communicates this confidence in its prediction (e.g. *This is an apple*). If the score goes from 0.5 to 0.8, then the "recognition confidence is relatively high, however the robot is not certain about its current interpretation" (Skočaj et al., 2009, p. 4) and the system communicates the lack of confidence (e.g. *I think this is an apple*).

As can be observed in Figure 10, both strategies leave the user the option to unlearn (Skočaj et al., 2009), that is, if the recognition the system has made of the object presented is wrong, having the possibility to correct it and, therefore, it will take an image of the object that it tried to recognise, assign it the correct label and attempt to *update* the system knowledge with the taken image in the correct category of the support set. Even with high confidence scores, this recognition may be wrong with categories which are visually very close (e.g. *boot vs. shoe*, or *pen vs. marker*). If the recognition is right, in both cases the system considers that it does not need to update the support set of images or to retrain the matching network model. It takes the image of the object that is seeing and saves it into the dataset.

For lower confidence scores, the system shows less confidence in its answers and the strategy for training changes as well. If the score is between 0.3 and 0.5, it is relatively low and therefore the system communi-

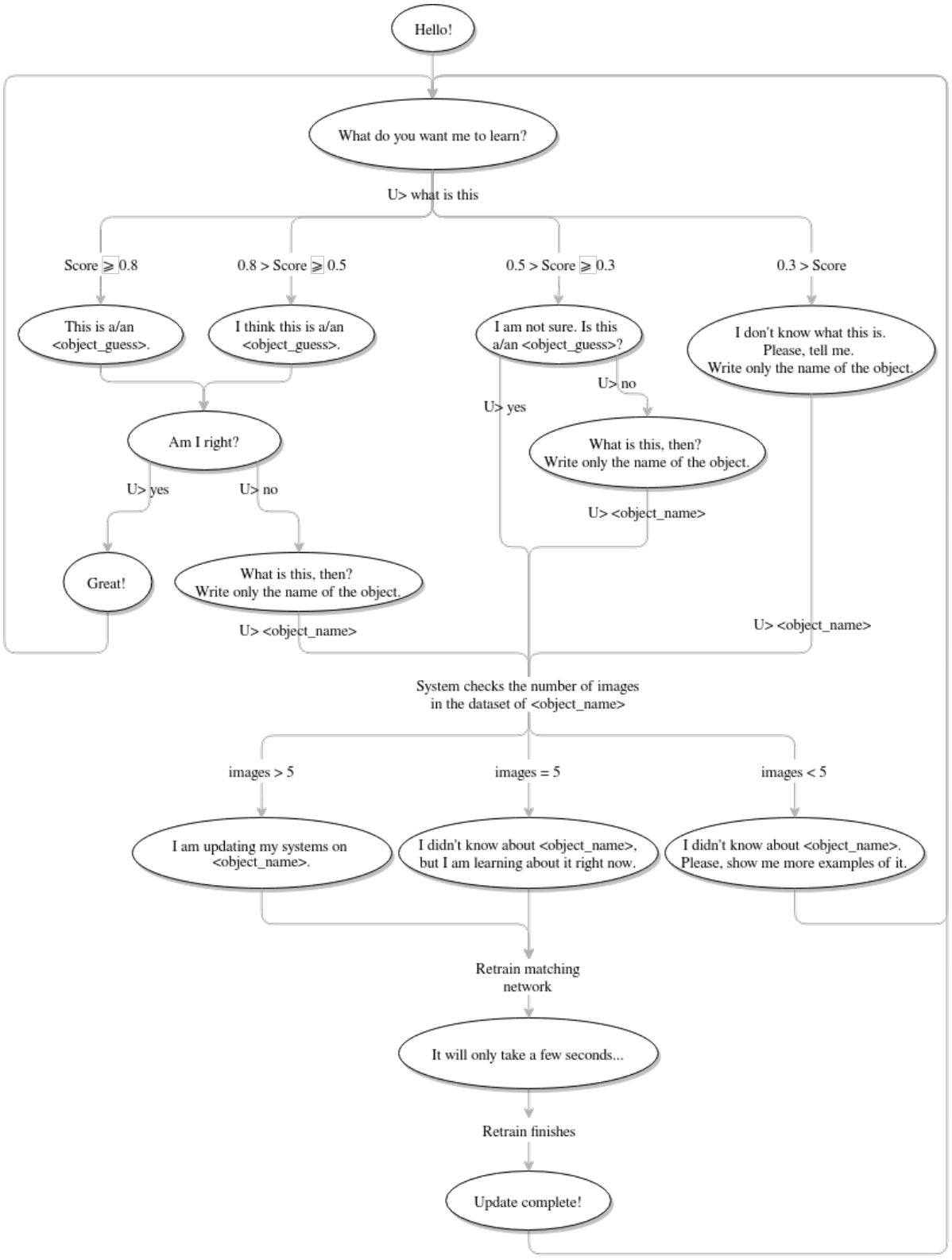

Figure 10: Interaction diagram for the user query *What is this?*. User utterances are marked with *U>*.

cates doubt about the recognised category and ask the user directly for clarification (e.g. *I am not sure. Is this an apple?*). Here the user can say that the recognition is correct or amend the category with the right one, as it is done when having more confidence. However, in both cases the system attempts to *update* the support set and retrain the model.

Lastly, if the confidence score is lower than 0.3, that means that the robot has not managed to classify the object correctly, and therefore it does not specify any category in its answer (*I don't know what is this. Please, tell me.*). In this case, all the scores for all the categories tend to be very close and therefore the recognition is not reliable. The system needs to query the user about the correct name of the object. Once the robot has the right category, it attempts to *update* its support set and retrain the matching network.

As shown in Figure 10, in all of the cases described above that end up in updating the support set and the matching network, the system always checks if it has enough images to learn about the category specified by the user (as it is done in the first interaction strategy of Figure 9). It could be, therefore, that the user asks about an object whose category the robot has not learned beforehand (for instance, because the user did not know). Then, the system would tell the user that it did not know about that object if it does not have five or more images of the object in the dataset (e.g. *I didn't know about apple. Please, show me more examples of it.*); tell the user that it did not know, but will *learn the new category* in case it has five images of that category in the dataset (e.g. *I didn't know about apple, but I am learning about it right now.*); or *update* its knowledge of the category if the robot already had the category in its support set (e.g. *I am updating my systems on apple.*).

4 Experiments and discussion

4.1 Experiment 1: System validation

For this first evaluation of our model we use the Omniglot dataset (Lake et al., 2015), which consists of 1623 grayscale images that represent characters from 50 different alphabets (Figure 11). Each of the categories in this dataset contains 20 images of the same character hand drawn by different people, which makes this dataset ideal for evaluating few-shot learning tasks.

The main objective of this experiment is to compare our matching network performance on Omniglot with the model in (Vinyals et al., 2016). It is expected that both models have a similar performance since our matching network is built taking the former as a reference. Additionally, we want to evaluate the general performance on this dataset and the impact on performance and the time taken for training our matching network with different number of labels (5-way and 20-way) and images per label (1-shot, 5-shot and 10-shot).

Since the images in Omniglot are grayscale, deep neural networks such as VGG16 pre-trained on colour images are not suitable for this dataset. Therefore, we have built a simple CNN encoder similar to the one used in (Vinyals et al., 2016, p. 6): "consisting of a stack of modules, each of which is a 3 × 3 convolution with 64 filters followed by batch normalisation, a Relu non-linearity and 2 × 2 max-pooling. We resized all the images to 28 × 28 so that, when we stack 4 modules, the resulting feature map is 1 × 1 × 64, resulting in our embedding function *f(x)*". For training this encoder, we use all the images from the Omniglot *background* split (19280 images divided in 964 different characters, each one being a category).

After training the encoder, we take the resulting pre-trained CNN layers to encode the images (similar to what it is done in the robot setup with the VGG16 CNN layers). For testing the matching network, we use the Omniglot *evaluation* split, which consists of 659 different characters that are not in the *background* split, so we know for sure that the few-shot learned categories have never been seen by any of the modules of our network.

We use the training strategy that we use in our robot for this experiment (defined in Section 3.4.2). From the *evaluation* split, a support set (S) is taken to train the matching network layers as both support and target. S has n labels (5-way or 20-way) and k images per each label (1-shot, 5-shot or 10-shot), as seen in Table 1. The rest of the images of the same labels are taken as target (t) to evaluate the classification process (that

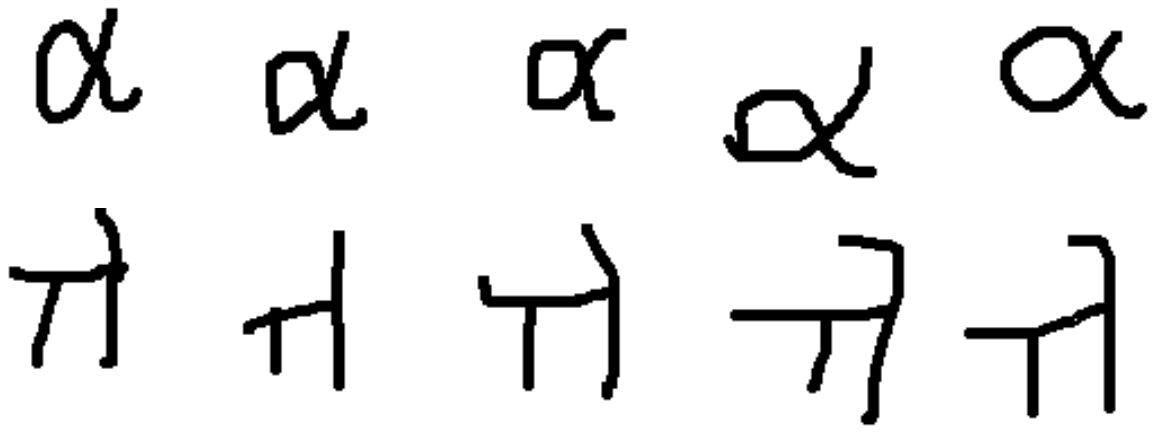

Figure 11: Ten samples from two characters of the Omniglot dataset.

Model	Metric	5-way			20-way		
		1-shot	5-shot	10-shot	1-shot	5-shot	10-shot
MN (ours)	Accuracy	96.0%	99.0%	99.0%	71.0%	93.0%	98.0%
	Encode time	0.94s	0.86s	0.98s	0.83s	0.81s	0.82s
	Training time	1.40s	3.77s	6.76s	3.25s	11.81s	22.59s
Vinyals et al. (2016)	Accuracy	98.1%	98.9%		93.8%	98.7%	

Table 1: Omniglot experiments. The first column on the left marks which version of the Matching Networks (MN) are the results from: the one in (Vinyals et al., 2016, p. 5) or ours.

is, 10 images per label).

We show in Table 1 the results of the experiment. Accuracy is almost the same across the different number of shots in 5-way, but with more categories it is more clear that (Vinyals et al., 2016) matching network performs better. However, we do not know which categories of the Omniglot *evaluation* dataset they used to evaluate their models and we have observed that evaluating different categories changes the results a lot, specially for 1-shot.

We can also observe that having at least 5 images clearly helps the model perform way better, but it also increases the training time significantly, specially as more labels are added to the model. It is also clear that the encoding of the images is really fast.

4.2 Experiment 2: Transfer Learning and Training Strategy

In the second experiment we use miniImageNet, an ImageNet split of 60,000 images distributed equally in 100 categories (600 per category), which makes this dataset more suitable for "rapid prototyping and experimentation" (Vinyals et al., 2016, p. 6) than the full dataset. Since the categories used in (Vinyals et al., 2016) were not released with their dataset, our splits are the ones proposed in (Ravi & Larochelle, 2017)[7]. These 100 categories are divided into three splits: 64 for training (*train* split), 16 for validation (*val*) and 20 for testing (*test*).

In this experiment we will evaluate two different VGG16 encoders.

Firstly, we pre-train a VGG16 encoder with all the images from miniImageNet *train+val* splits (48000 images in all, 80 categories). We have performed this pre-training in 100 epochs and with a batch size of 64 images. Then, we save the encoding layers for encoding the images later. This setting replicates the pre-training of the encoder layers that Vinyals et al. (2016) perform in their experiments, so we can compare our results with the performance of their model.

Secondly, we take the VGG16 CNN model already pre-trained on the entire ImageNet, which is the same model downloaded from Keras and that we use for in the robot enviroment (as explained in Section 3.2.1). In this setting, we are using transferred weights from a dataset which is much larger than miniImageNet, so we can compare how much it benefits from having an encoder pre-trained on a large dataset.

In this experiment we are also evaluating the two training strategies described in Section 3.4 in our matching network: the original one (Vinyals et al., 2016), and the one designed for our robot.

[7] https://github.com/twitter/meta-learning-lstm

Figure 12: Ten samples from two labels of the miniImageNet dataset.

Firstly, we replicate the strategy of (Vinyals et al., 2016). As explained in Section 3.4.1: we take k images (1-shot, 5-shot or 10-shot) of n categories (5-way or 20-way) of the *train+val* split of miniImageNet to build the support set (S) that we use to train the matching network. We use the rest of the images of the same categories as target (t) for training the matching network. We then retrain our model on a different batch of n categories and k images per category. We do the same continuous training until we have trained our model on all the categories of the miniImageNet *train+val*.

After training, we evaluate the classification on a support set of the same size built with the *test* split of miniImageNet, which are completely new categories not seen during the training stage. The rest of the images of the same categories of the *test* split (590 images per category) are used as target to measure the classification accuracy.

Secondly, we use the same training strategy that we use in our robot enviroment as described in Section 3.4.2. In the training phase, we build a support set (S) from the *test* categories which has k images (1-shot, 5-shot or 10-shot) of n categories (5-way or 20-way) and train our matching network using S as both support and target. After that, in the evaluation phase we take the same S as support set and measure the accuracy of the model using as target for classifying the rest of the images of the same n categories (590 per category) that we used to build S.

Comparing these two training strategies we can check if it is really necessary to train our matching network to differentiate the labels that we want to classify, or if we can pre-train it in other labels and transfer this knowledge to classify the *test* labels.

Therefore, we have four different settings for this experiment:

- VGG16 pre-trained by us on miniImageNet *train+val* and using the training strategy in (Vinyals et al., 2016).
- VGG16 pre-trained on the entire ImageNet (Keras model) and using the training strategy in (Vinyals et al., 2016).
- VGG16 pre-trained by us on miniImageNet *train+val* and using our training strategy.
- VGG16 pre-trained on the entire ImageNet (Keras model) and using our training strategy.

In Table 2 we can observe that the best performing setting by far is the one that uses the VGG16 encoder

VGG16 set	Training strategy	Metric	5-way			20-way		
			1-shot	5-shot	10-shot	1-shot	5-shot	10-shot
miniImageNet	**Vinyals**	Acc	20.2%	22.2%	24.6%	7.7%	9.8%	11.4%
ImageNet	**Vinyals**	Acc	29.7%	44.1%	43.9%	12.3%	25.1%	27.2%
miniImageNet	**Ours**	Acc	35.1%	48.0%	49.8%	13.4%	22.4%	26.9%
ImageNet	**Ours**	Acc	**75.8%**	**89.8%**	**98.8%**	**52.5%**	**74.2%**	**82.6%**
All	All	Enc time	1.20s	1.56s	2.02s	1.34s	2.11s	2.42s
All	**Vinyals**	Train time	241s	296s	343s	202s	260s	313s
All	**Ours**	Train time	1.41s	4.06s	7.50s	3.50s	13.70s	24.50s
Vinyals et al. (2016)		Acc	46.6%	60%				

Table 2: The first column at the left specifies the dataset in which the VGG16 encoding layers are trained. The second one specifies the training strategy, where *Vinyals* is the one used in (Vinyals et al., 2016) (described in Section 3.4.1) and *ours* is the one used in our robot enviroment (Section 3.4.2). We also keep track in the last rows of the time that the system needs to encode the images and to train the matching network. Finally, in the bottom we show the results reported in (Vinyals et al., 2016) for comparison.

pre-trained on the largest dataset (ImageNet) and the training strategy used in our robot, which is higher than the results in (Vinyals et al., 2016). However, we cannot compare with more categories since (Vinyals et al., 2016) does not show results for 20-way models.

Additionally, it is clear that for our model, our training strategy demonstrates to be more effective in performance than the Vinyals one. However, that is expected because Vinyals strategy uses learning transferred from other categories to evaluate new ones, while our strategy trains a model on the same categories that we are using in evaluation. However, since training is demonstrated to be fast with our strategy (as shown in Table 2) and performance is also much better, our strategy is more convenient for our robot object recognition module.

We can also observe in Table 2 that the models that we have trained and that use (Vinyals et al., 2016) training strategy (the first two rows) perform worse in our model than in the one in the original paper, even with the weights trained on the entire ImageNet. Since the difference between both Matching Networks is that we have simplified the embedding functions by removing the BiLSTM (in the g function for S set) and the LSTM (in the f function for t set), we can hypothesise that the drop in performance is probably caused by this difference.

However, it is also a possibility that the drop in perfomance is caused by the fact that we do not know what training parameters are used to train the original model, since (Vinyals et al., 2016) does not report them. For instance, what loss function, optimizer, learning rate, number of epochs or batch size are used for training their matching network model.

It is also clear that having VGG16 pre-trained on the largest dataset (ImageNet) is better in order to take full advantage of learning transferred from another dataset. No matter the training strategy that we use, accuracy improves a lot in all the cases in which the image encoder is trained on ImageNet.

4.3 Experiment 3: Baseline evaluation

In this experiment we use images from the robot's camera (SOTA dataset, described in Section 3.3) with which we can simulate object recognition of the robot without any interaction strategy which therefore serves as a baseline for further work.

VGG16 set	Metric	5-way			20-way		
		1-shot	5-shot	10-shot	1-shot	5-shot	10-shot
SOTA	Accuracy	38.0%	56.0%	56.0%	9.0%	13.5%	22.5%
	Encode time	1.11s	1.51s	2.12s	1.45s	1.94s	2.57s
	Training time	1.45s	4.03s	7.07s	3.15s	12.46s	26.13s
ImageNet	Accuracy	**66.0%**	**90.0%**	**94.0%**	**41.5%**	**71.0%**	**86.5%**
	Encode time	1.12s	1.63s	2.15s	1.41s	1.93s	2.39s
	Training time	1.43s	3.57s	7.27s	3.26	12.15s	25.99s

Table 3: Experiments on SOTA. The first column on the left marks the dataset used to train the VGG16 encoding layers.

We use two differently trained VGG16 encoding layers in this experiment. Firstly, we take the SOTA dataset (400 images) and pre-train VGG16 with it. Secondly, we compare it with the encoder pre-trained on ImageNet to observe how is the robot benefitted from being trained on a large dataset instead of using the same images of its domain for training the encoding layers.

For both settings we use the same training strategy. That is, the one used in our robot enviroment (Section 3.4). As in the previous experiments, we take a support set (S) with n labels (5-way or 20-way) and k images per label (1-shot, 5-shot or 10-shot) and use it as support and target sets for training, and as support set for the evaluation phase. As the target set for evaluating the model, we take the remaining 10 images per category. That is, the ones that have not been used to train the model.

Results in Table 3 show that, even with the VGG16 pre-trained on the same images that we use to evaluate the model, it still performs much better when it is pre-trained in a large dataset. Therefore, we can definitely conclude that it is clearly beneficial to have the transfer learning from large datasets in the image encoder module, even more than having an image encoder specialised in our domain. However, it is also worth noting that SOTA is too small to train an effective image encoder on it.

This experiment is also useful to measure the baseline performance of our model in the robot domain, since all the images in SOTA were taken with the robot camera. Although 1-shot per category would be an ideal setting in terms of the time performance, results demonstrate that we need more images per category to achieve good recognition performance. As we see these results and the ones in the experiments in Sections 4.1 and 4.2, five images per category seems the optimal setting for achieving good performance with reasonable training times. Therefore, it is the one that we use for our robot.

We have to remark that in this experiment we have not applied any selection of the support dataset. That is, we have taken images from SOTA at random to build S with them (but a fixed selection of images so we can reproduce the experiment with the same S). Different selections of a support set might improve or worsen the performance of the system.

We also did not test the perfomance on the same or different objects of the same category (de Graaf (2016) does it). Maybe for each object of a category in the dataset it is only needed one image sample in the support set, but in the case of our SOTA dataset some categories only have very few different objects, and we would need at least five different objects per category to run that experiment. It would be worth comparing the performance on a dataset where there were images of different objects of the same categories. Implementing a way to control the selection of the support set would be a great asset for our robot.

4.4 Experiment 4: Building a support set with images from another domain

In this experiment we investigate how well our matching network is performing when taking images from another domain. These images from the same labels but taken with other cameras (e.g. images from online datasets taken in different situations and eviroments). The objective is to demonstrate that our framework can benefit from taking images of the same categories but from other domains and pre-training our model on these.

We have collected a small dataset, which we will call SOTA-external, with images extracted from the Internet, specially from ImageNet or Flickr. These images depict an object of the same 20 labels available in our SOTA dataset, so we can build a support set with up to five images per label (100 images overall). Due to copyright issues, SOTA-external cannot be published.

In this experiment we keep the VGG16 weights trained on the entire ImageNet, since we have demonstrated that this is the most favorable setup. The focus here is in the building of the support dataset and the training strategy itself.

We have devised two different training strategies for this experiment. Firstly, we build a support set (S) with the images from SOTA-external and use S as both support and target to train our matching network. In the second scenario, the support set for training is the same SOTA-external data, but the target set are the images from SOTA. For evaluation, the support set is always SOTA-external and the target are the images from the SOTA dataset because those are the objects that our robot is trying to recognise and are taken with its camera.

These two scenarios are performed to test if it is sufficient to train our model using the data from another domain, or if it is better to have images from our domain to compare them with the ones from the SOTA-external. These results will be compared with the baseline established in the previous experiment.

As shown in Table 4, performance with few categories is almost equal in both settings, but with more categories it is clear that using the images from our robot domain (SOTA) as t for training is clearly better than using the images from the Internet domain (SOTA-external). Comparing the performance of the second setting with the baseline results (Table 3), the performance of the second setting is better than the baseline (except in 5-way 5-shot).

It seems possible to use images from other domains as support set in our robot scenario without having images that could be used as target to train. Although there is a gap in performance, the system still performs acceptably. However, the first scenario is only a baseline: it performs well but also shows that domain adaptation is necessary. That is, the robot must learn a locally adapted model. The second scenario shows that transferred learning from another domain helps in comparison to domain-specific learning only.

For instance, if there is a scenario with a new room in a house in which the robot has not been trained (e.g.

Train support set	**Train target set**	**Metric**	**5-way**		**20-way**	
			1-shot	**5-shot**	**1-shot**	**5-shot**
Internet data	**Internet data**	Accuracy	71.1%	**91.1%**	39.4%	60.0%
Internet data	**SOTA**	Accuracy	**75.6%**	86.7%	**53.3%**	**73.9%**

Table 4: Experiments with images from another domain. The first row shows the accuracy of the model trained using the SOTA-external dataset as both support and target set, while the second row shows the model trained using SOTA-external as support and our dataset as target.

the kitchen), the robot could have pre-learned categories for common objects in the kitchen (e.g. plate, spoon, cup, glass, etc.). Then, we can improve this knowledge with new images and labels taken in the actual scenario with the user's robot camera.

However, it is worth remarking that these experiments have been performed with handpicked images that show one or two objects of the same category as the ones available in SOTA. In a real scenario with a robot, selecting the images from another domain as support data would require a labeled corpus that contains the same kind of images (such as ImageNet).

External corpora may be useful also to learn new categories, since we can complement the inputs from the robot camera with images extracted from another domain to learn new categories efficiently from the first input.

4.5 Experiment 5: Learning new classes of objects

The objective of this experiment is to test how the system learns new labels. That is, we want to know how many images are needed for our robot to recognise efficiently a new label and therefore what are the implications of learning new labels for modelling interaction strategies of a robot.

We have designed a simulation of this learning process for each of the labels in SOTA, as shown in Figure 13. We simulate the learning process by training matching networks on 19 labels with five images each, which represent the categories that the robot already knows, and then adding the remaining label in SOTA to the support set which is learned from 1 to 5 images and represents a newly learned category.

For instance, for the first column *apple* we train a matching network model on a support set S built with the other 19 labels of SOTA plus the 1-shot learned *apple*. We also train four more models with 2-shot, 3-shot, 4-shot and 5-shot learned *apple*. S is also used as target for training, as it is done our robot training strategy (Section 3.4.2).

In the evaluation phase we take the same S set and test the recognition accuracy of *apple* on the remaining 15 images of the same label *apple* for each of the models, which are the t images for the model to classify. We repeat that same procedure until we have the five models evaluated for each new label.

Figure 13 shows that four to five images are necessary for most of the labels to have a reliable object recognition. Also, some labels are clearly easier to learn than others. For instance, *bottle*, *box*, *clothespin* and *marker* do not reach more than 40% accuracy in the best case, while *apple*, *book* and *stuffed toy* did not need more than three images to get to about 80% accuracy.

Perhaps needing 4 or 5 images for most of the newly learned labels for an effective learning might be due also to the negative effect of not balancing the support set, since all the other labels have 5 images each. Further work should validate what happens if we increase to 10 shots and try more unbalanced sets.

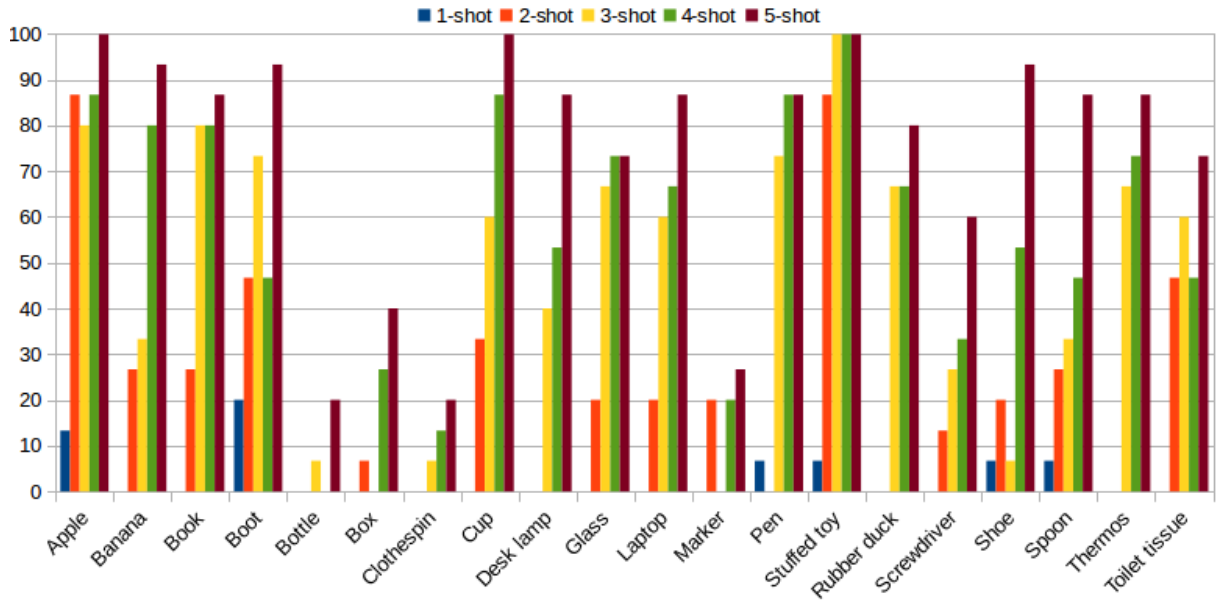

Figure 13: Accuracy of learning new labels specified on the x axis. Each of the bars represents the accuracy of the classification of 1-shot (left) to 5-shot (right) training.

The plot also shows some irregular ups and downs in accuracy when adding new images to some labels. For instance, the accuracy of the 2-shot learned *boot* is 46.7%, 3-shot goes up to 73.3%, but 4-shot goes down again to 46.7%. These irregularities could be explained by taking into account that some images added to the previous may confuse more the model. For instance, images with different objects or having strange points of view of the object. They could be also produced by the effect of the background in the images of the objects, which is not filtered by the system.

5 Conclusions and further work

In this work we have presented an evaluation of few-shot learning methods with neural networks in relation to an interactive agent such as the KILLE framework (de Graaf, 2016). The researched frameworks for achieving this goal are transferred learning and Matching Networks (Vinyals et al., 2016), which explores a way to train a neural network to do one-shot learning. That research has allowed our system to learn categories of objects fast and needing very few samples.

As shown by our results (Section 4), we have demonstrated that fast learning with neural networks is possible and reasonably effective with few categories and therefore suitable for interactive situated agents. Baseline results (Sections 4.3 and also 4.5) show that the accuracy is quite high taking into account the low training times. That is, training a model for classifying 20 categories of objects with only 5 images per category in about 15 seconds and achieving an accuracy of 71% seems an encouraging result.

Of course, the fact that the training is performed during the interaction and the user has to stop until a new model is trained is not good as a user experience. Since our focus was on the performance of neural networks in a robot scenario, we have not dealt with this. However, as remarked in Section 3.4, this could be avoided in two ways. The most basic would be to configure the system so it only trains a model when a user is not interacting. It would also be possible to train the matching networks parallelly to the interaction (that is, without needing to interrupt it) and while the model is being trained the system would still use the old model. The evaluation of these strategies by human subjects should be part of our future work.

The second way would be to use learning transferred from a large datasets as in (Vinyals et al., 2016). Their matching network is trained in different categories from the ones which are tested to recognise, so they demonstrate that it is possible to make use of that knowledge from other datasets.

The results of the experiments in Sections 4.2 (Table 2) and 4.4 (Table 4) demonstrate that it is possible to use transfer learning in our matching network, but it depends at which level the transfer is done. We can transfer knowledge effectively from different domains as long as we take the same categories that we want to recognise with the robot. That is, the ones that will be in the support set when evaluating.

However, our matching network is not capable of using the knowledge transferred from different categories efficiently. The data used to train our model in Experiment 2 is very different from the data in the support set when recognising new objects since the categories are not the same. As it is remarked in (Yosinski et al., 2014), having very pre-trained a model on very different data from the target task decreases the benefit of using transferred learning. Further work should investigate how it would be possible to take full advantage this more distant type of transferred knowledge from new categories.

Contrary to the matching network module, the VGG16 CNN layers contribute with transfer learning for the image encoding module, and we have demonstrated that it is very efficient in our models. In the original matching networks (Vinyals et al., 2016), the CNNs were trained along with the embedding functions, but we have skipped this training stage for the CNNs by using the knowledge transferred in the models trained on ImageNet.

Further work with Matching Networks in this robot framework should also investigate the influence of different support sets on the improvement of the system. As remarked in Section 4.3, having mechanisms to control better the selection of the support dataset would most probably make the system more efficient. For instance, maybe for the system to learn the label *apple* it only needs a sample image of a green apple and a red apple in order to be able to recognise both kinds of *apple* (in Figure 13, *apple* only needs two images to recognise efficiently). Other categories are more difficult to learn since they have more different objects (e.g. there are several different types of *box*) or the objects can vary a lot from different perspectives (e.g.

a *shoe* can be very different visually when observed from the side *vs.* when looked from the front).

Another open question regarding our robotic framework is whether to use background filtering as in (de Graaf, 2016). Not using it allows us to avoid its shortcomings (detailed in Section 3.1), but it also allows the background of the objects in the image to influence negatively the recognition of objects. A recommended extension to our work would be to investigate how to implement ways to reduce the impact of the background, maybe as a different module which could be connected with the existing ones.

For instance, Girshick et al. (2013) propose a R-CNN model (Regions with CNN features) which is capable of localising and segmenting relevant regions of an image in different bounding boxes, each containing an object. Another procedure for background filtering improvement could be implementing models with attention over the objects of the image, such as those in Xu et al. (2015) and Lu et al. (2016).

A module that should be a focus of considerable further work is the dialogue module. Since it was not the main focus of this research, dialogue management in our work is a functional module implemented in Python, very simple, it supports text and it is state-based (see Section 3.5). Further work should investigate an implementation of a more sophisticated dialogue manager module, maybe an extension of the *OpenDial* dialogue manager as was implemented in (de Graaf, 2016) or another dialogue manager with voice support that can be used with Robot Operating System (ROS).

Furthermore, interaction strategies could also be improved. For instance, as remarked in Section 2.3, our system should be able to *unlearn* efficiently uncorrectly learned labels. Also, interaction strategies used in other robot frameworks could be investigated as a possible way to make the interactive learning of our framework more efficient, which would help to make a more effective robot for object recognition.

Finally, our work has focused on learning objects, but we have not looked at learning spatial relations between objects, which is a capability implemented in the original KILLE robot (de Graaf, 2016). Further studies in this framework could examine the implementation of spatial relations learning with our neural network setup with few shot learning for object recognition.

References

Bradski, G. & Kaehler, A. (2008). *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, Inc.

Cai, Q., Pan, Y., Yao, T., Yan, C., & Mei, T. (2018). Memory matching networks for one-shot image recognition. *CoRR*, abs/1804.08281.

Cano, J. M., Dobnik, S., & Ghanimifard, M. (2019). Interactive visual grounding with neural networks. In *Proceedings of the 23rd Workshop on the Semantics and Pragmatics of Dialogue - Poster Abstracts* London, United Kingdom: SemDial.

de Graaf, E. W. (2016). *Learning objects and spatial relations with Kinect*. Gothenburg, Sweden, June, 8th 2016. Supervisor: Simon Dobnik, examiner: Richard Johansson, opponent: Lorena Llozhi: Master's thesis, Department of Philosophy, Linguistics and Theory of Science. University of Gothenburg.

Deng, J., Krause, J., Berg, A. C., & Fei-Fei, L. (2012). Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3450–3457).

Dobnik, S. & de Graaf, E. (2017a). Kille: a framework for situated agents for learning language through interaction. In *Proceedings of the 21st Nordic Conference on Computational Linguistics* (pp. 162–171). Gothenburg, Sweden: Association for Computational Linguistics.

Dobnik, S. & de Graaf, E. W. (2017b). Kille: learning grounded language through interaction. In *Proceedings of the Workshop on Formal Approaches to the Dynamics of Linguistic Interaction 2017*: CEUR-WS.org, urn:nbn:de:0074-1863-7.

Dosovitskiy, A., Springenberg, J. T., Riedmiller, M. A., & Brox, T. (2014). Discriminative unsupervised feature learning with convolutional neural networks. *CoRR*, abs/1406.6909.

Girshick, R. B., Donahue, J., Darrell, T., & Malik, J. (2013). Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524.

Harnad, S. (1990). The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1), 335 – 346.

Koch, G., Zemel, R., & Salakhutdinov, R. (2015). Siamese neural networks for one-shot image recognition. In *Proceedings of the 32 nd International Conference on Machine Learning*.

Lake, B. M., Salakhutdinov, R., & Tenenbaum, J. B. (2015). Human-level concept learning through probabilistic program induction. *Science*, 350(6266), 1332–1338.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312.

Lison, P. (2014). Structured probabilistic modelling for dialogue management. *PhD thesis*, University of Oslo.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99 (pp. 1150–1157). Washington, DC, USA: IEEE Computer Society.

Lu, J., Xiong, C., Parikh, D., & Socher, R. (2016). Knowing when to look: Adaptive attention via A visual sentinel for image captioning. *CoRR*, abs/1612.01887.

Ordonez, V., Deng, J., Choi, Y., Berg, A. C., & Berg, T. L. (2013). From large scale image categorization to entry-level categories. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13 (pp. 2768–2775). Washington, DC, USA: IEEE Computer Society.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., & Y. Ng, A. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Ravi, S. & Larochelle, H. (2017). Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*.

Roy, D. (2002). Learning visually-grounded words and syntax for a scene description task. *Computer Speech and Language*, 16, 353–385.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3), 211–252.

Santoro, A., Bartunov, S., Botvinick, M., Wierstra, D., & Lillicrap, T. (2016). Meta-learning with memory-augmented neural networks. In M. F. Balcan & K. Q. Weinberger (Eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research* (pp. 1842–1850). New York, New York, USA: PMLR.

Schütte, N., Kelleher, J., & Mac Namee, B. (2015). Reformulation strategies of repeated references in the context of robot perception errors in situated dialogue.

Simonyan, K. & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv e-prints*, (pp. arXiv:1409.1556).

Skočaj, D., Janiček, M., Kristan, M., Kruijff, G.-J. M., Leonardis, A., Lison, P., Vrečko, A., & Zillich, M. (2010). A basic cognitive system for interactive continuous learning of visual concepts. In *ICRA 2010 workshop ICAIR - Interactive Communication for Autonomous Intelligent Robots* (pp. 30–36). Anchorage, AK, USA.

Skočaj, D., Kristan, M., & Leonardis, A. (2009). Formalization of different learning strategies in a continuous learning framework. In *Proceedings of the Ninth International Conference on Epigenetic Robotics; Modeling Cognitive Development in Robotic Systems* (pp. 153–160).: Lund University Cognitive Studies.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., & Wierstra, D. (2016). Matching Networks for One Shot Learning. *arXiv e-prints*, (pp. arXiv:1606.04080).

Wang, Y. & Hebert, M. (2016). Learning to learn: Model regression networks for easy small sample learning. In *European Conference on Computer Vision (ECCV)*.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A. C., Salakhutdinov, R., Zemel, R. S., & Bengio, Y. (2015). Show, attend and tell: Neural image caption generation with visual attention. *CoRR*, abs/1502.03044.

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks? *CoRR*, abs/1411.1792.

6 Appendices

6.1 Robot implementation

As stated in Section 3.1, our robot implementation consists of three Python modules: `dialogue.py`, `matchingnets.py` and `recognise.py`. The main reason for implementing them as separated modules is to allow the possibility of exchanging in the future some modules by others or even add more if seen convenient without needing to change a lot the other modules.

We have set up a public GitHub[8] repository so anyone can have access to the code of this project and the SOTA dataset (see Appendix 6.2). Technical instructions on how to install the necessary software and run the project are also provided in the repository.

6.1.1 dialogue.py

This module takes care of the dialogue management for the robot. As stated in Section 3.5, this is finite state and text-based. The different states are represented inside the code as Python functions inside the `DialogueManagement` object.

This object takes care of printing the different utterances from the system and process the utterances from the user. Based on this answers, it communicates to `recognise.py` the actions that should be performed with the robot camera and the model according to the user utterances. It also sends the labels of the objects that the user has said to classify the inputs from the camera.

Additionally, it receives the answers from the system to advance to further functions (states) inside the interaction.

6.1.2 matchingnets.py

This module contains the neural network model (inside the object `MatchingNets`) and some functions for collecting the images needed for running a session with the robot.

The `MatchingNets` object contains the architecture of the model: the VGG16 model ready to encode images (function `vgg16_encoding`) and the matching network (function `model_layers`). It also contains the function to run the model for training (`run_model`) and the function to predict on a set of images and return the predictions and the accuracy of the same predictions.

Our own implementation of Matching Networks has been done by analysing Vinyals et al. (2016) paper and two implemented interpretations of the architecture: one made using the TensorFlow library[9], and another that uses the Keras library[10].

6.1.3 recognise.py

This module handles the image inputs from the robot camera, runs the model in `matchingnets.py` when retraining it is necessary, receives the user inputs from `dialogue.py` and sends the system inputs to continue with the interaction.

[8] https://github.com/jcanosan/Interactive-robot-with-neural-networks
[9] https://github.com/markdtw/matching-networks
[10] https://github.com/cnichkawde/MatchingNetwork

This script takes care of receiving the images of the camera from the ROS node `/camera/rgb/image_color` (lines 92-93 in the code). The function `image_callback` (lines 129-199) receive the different frames from the ROS node and uses them for the classification. To save the images into the corpus, we use the function `save_img` (lines 244-264), which is called when the user presents an object to the camera (e.g. *This is a rubber duck*) or when the system needs to update their knowledge or learn a new category.

6.2 SOTA dataset collection

SOTA (Small Objects daTAset) is a dataset of 400 images distributed equally into 20 categories (Section 3.3). The images were taken using the same Kinect v1 RGB camera while we were testing the different components of the scripts and the interaction. The resolution of the camera is 640×480 pixels, although the images taken are resized to 224x224 pixels since it is the default size that VGG16 and most deep image convolutional neural networks use by default. Using the pre-trained VGG16 model would not have been possible with another image size.

The camera is connected to our implementation making use of Robot Operating System (ROS) inside the `recognise.py` module. The images portrait a single object of interest which is normally centered in the image. Objects are often rotated, specially in the categories with less variety of objects (e.g. *apple*, *banana* or *rubber duck*). They also often have different backgrounds sorrounding the objects, since a real use case scenario would have background noise that can interfere with the object recognition process. It is also good to see how the background influences in the object recognition and, in further work, use the same dataset to observe how background filtering techniques (mentioned in Section 5) can make better the recognition process compared with not filtering the background.

The collection of these images was intentended to have a dataset ready for using in our robot domain as the support set of the Matching Networks module for both interacting with the robot by using it to build support sets and evaluating the same neural network performance. However, the images can also be used in other domains with other datasets.

The scope of the dataset is to portrait common objects that could be normally present inside any house. Although most of the categories portrait objects available in a bedroom scenario (e.g. *stuffed toy*, *shoe*, *pen* or *book*), some contain objects that maybe you could find better in other rooms (e.g. *spoon*, *apple*, *cup* or *rubber duck*). However, it is not improbable to find objects of those categories in a room as well (maybe with the exception of *toilet tissue*).

However, although it requires further experimentation, it could be possible to bias the models that use our robot in a bedroom scenario to recognise the most common categories inside that room above the others, but leaving it the option to recognise objects that are not that common. For instance, it is very possible to find a *rubber duck* and a *spoon* in a room, although it seems less possible than finding a *shoe* or a *pen*. Therefore, we could introduce in the support set more images of the most common objects introducing less images of the rarest ones.

SOTA is available inside the same GitHub repository used for this project[11]. It is licensed under Creative Commons Attribution 4.0 International[12]. Anyone is free to share an adapt this dataset as long as appropriate credit is given to the original author.

[11] https://github.com/jcanosan/Interactive-robot-with-neural-networks/tree/master/utils/datasets
[12] https://creativecommons.org/licenses/by/4.0/